Analysis of MTA Bus-Time Data

Introduction:

Massive quantity of bus location data is pulled in through vehicle location systems. The project aims at improving the quality of information reported to the passengers using this data through the cell-phone application MTA Bus Time. The intention is to provide real-time estimates on bus arrivals to transit passengers. The same information is also used to aid the transportation agency to audit and enhance operational performance.

Why is the quality of predictions important?

- Transit passengers are typically attentive to the bus arrival times at the bus stop.
- Reporting accurate real-time predictions proves to be helpful for the transit passengers to plan their trips in an improved manner.
- The reduction in the amount of waiting time is a good measure of a superior customer experience.
- In situations where the passengers are misinformed, meaning, the bus does not arrive at/closer to the stated time on the cell-phone application – MTA Bus Time, the increased restlessness can lead to annoyance.
- Accurate arrival times also help at the operational level.
- The information can help operators learn whether buses are getting congested or dispersed.
- Accurate bus arrival data supports decision making that can improve service performance at both the operational and strategic levels.

Overview:

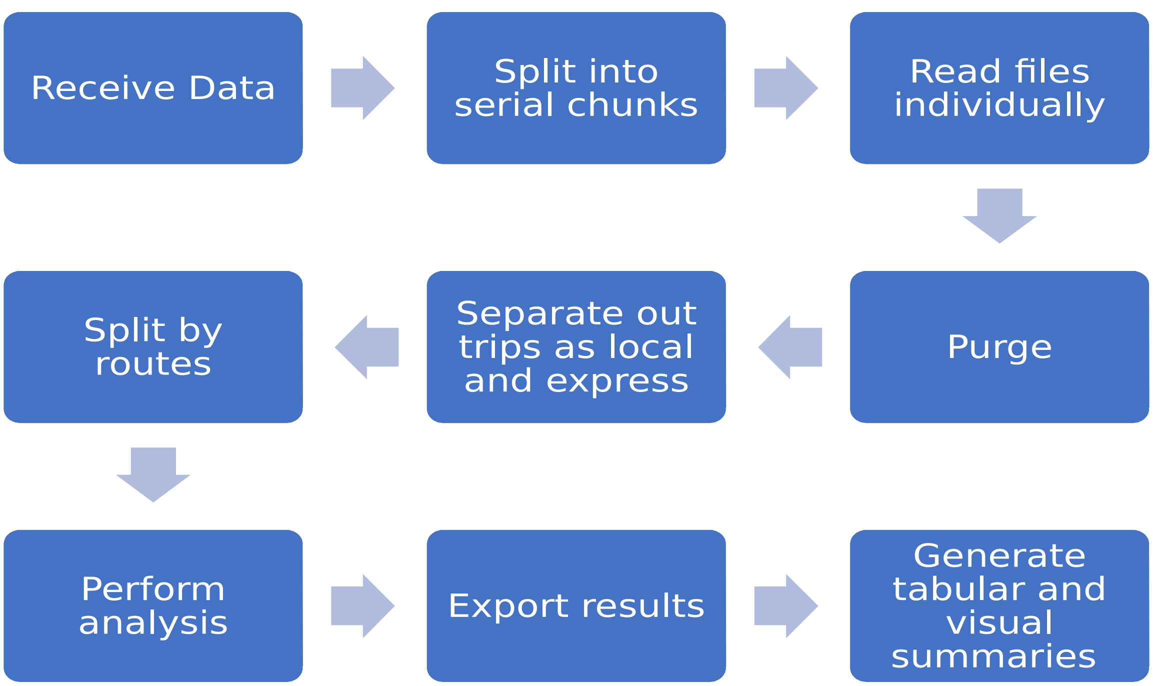


Fig. 1: Process flow of the project

Process flow:

- The data is obtained from the Server and stored as a CSV file.
- The raw file is then loaded into R-Studio, an open source integrated development environment for R (programming language for statistical computing and graphics).
- The reason being, the raw form of data needs a good amount of purging to transform its raw state to an analyzable form.
- Using various data cleaning methods, and filtering techniques, the data is prepared for analysis.
- Statistical methods are applied for estimating the optimum relationship between attributes responsible for the predicted arrival time of the bus which is reported to the passengers.
- Based on the newly obtained relationship, new bus time predictions are calculated and compared with the original ones using a summary table.
- The difference in the accuracy between the original and newly obtained predicted arrival times is compared using visualization tools, and meaningful insights are gained through such results.

Detailed Explanation:

1. System specifications:

The project has currently been done on a machine with the following specifications:

Operating system: Windows 10

Software: R-Studio Version 1.1.383, R version 3.4.2 (2017-09-28).

Installing R-Studio is straightforward. Prior to installing R-Studio, the only thing required to be installed on the machine is R.

Here is the webpage link for installing R and R-Studio: https://courses.edx.org/courses/UTAustinX/UT.7.01x/3T2014/56c5437b88fa43cf828bff5371c6a924/

The packages that would be required for smooth functioning of the R code, are all included as commands in the code itself. Therefore, explicit installation of these packages is not required. All the required packages will be installed in real-time when the code is executed.

This section does not imply that the above-mentioned specifications are the minimum requirement; the environment is quite flexible in terms of compatibility as far as this project is concerned.

2. Data collection:

The data is pulled in from the MongoDB server in the form of a comma delimited value file (.csv).

Depending on the size of the file, it can be read directly in Microsoft Excel or in R-Studio.

Case 1:

If the file contains less than 1,000,000 lines of information or in terms of size is less than 150MB, it can be directly read into MS Excel (if needed for checking the column names).

Case 2:

If it is a larger file that contains less than 2,000,000 lines of information or in terms of size is less than 655 MB, it can be read in R-Studio.

Case 3:

Files having more than 2,000,000 lines of information will not be efficiently read in R-Studio. The solution is to split these files into smaller chunks, where each of these chunks have a maximum of 2,000,000 lines.

Here are the steps to split the huge raw file into chunks:

For Windows Operating System:

Install bash by following the instructions given on this webpage:

https://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/

Once, the installation is complete, create a notepad file containing the script to achieve chunk-wise split.

The notepad file used can be found in the folder ‘scripts’, the file is named shellscript.txt.

It is important that this script file is stored in the same directory as the huge raw file.

Linux Operating System users originally use the script, however, to run the script in Windows Operating System, we carry out the above steps.

For Linux Operating System:

There is no requirement of installing bash, as the splitting operation can be carried out in the command line itself. The same script which can be found the file “shellscript.txt” can be used for obtaining multiple chunks out of the entire raw file.

Now that the huge raw file has been split into pieces of equal lines of records, all these pieces will appear in the same directory as the raw file and the script.

There’s another script for renaming the new chunks so that it becomes easier to read them one by one in R-Studio.

Please note, each of the newly obtained pieces will contain equal number of records and this data will still be raw, meaning, it still will not be in an analyzable form.

3. Exploring the Raw file:

3.1 Load Data:

It is necessary that the path to the raw file is fed in R-Studio. This path is the location where the file would be stored. It is also known as the work directory. All those files that are required to be read during the execution of the code should be stored at this location. Setting the work directory is accomplished in the very first line of the code.

E.g. setwd("F:/Prathamesh Shinge/Documents/New folder/test_linux")

The command ‘setwd’ stands for set work directory.

Now that the work directory has been set, let us look for the two available methods to read the raw file into R-Studio.

One way of reading the raw csv file would be using the command read.csv which needs no explicit installation of additional packages. This means, the following command would itself be enough to load the file into R-Studio: Raw_file = read.csv("split_aa.csv"). Here, the file that is loaded is named as ‘split_aa.csv’ which also happens to be the first of the many chunks formed. This file is stored as a data-frame which is named as Raw_file; the data frame can have a name of your choice. The concept of a **data frame** refers to "tabular" **data**: a **data** structure representing cases (rows), each of which consists of several observations or measurements (columns).

The second and comparatively faster way of reading the raw file would be the command read_csv, however, this one will only execute if the package ‘readr’ has been installed. The required sequence of execution would be,

install.packages(“readr”)

library(readr)

Raw_file = read_csv(split_aa.csv)

Not only is this method of reading faster, but also it displays the percentage of file being read in real time. Thus, one can view what amount of the file has been read and how much is left.

Note: The above sequence of commands is consistent throughout the code for installing a package and then using it from the library.

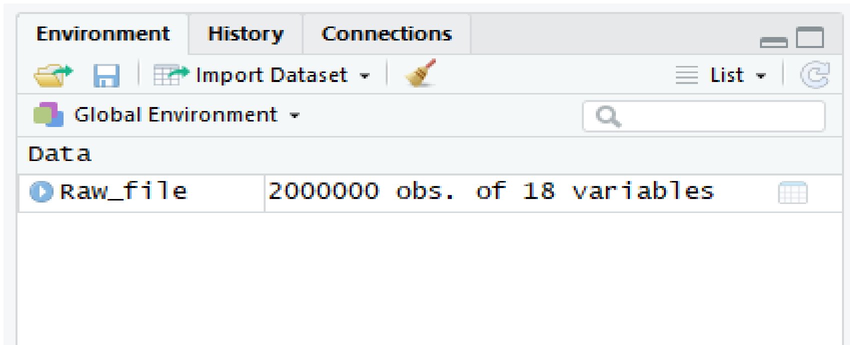
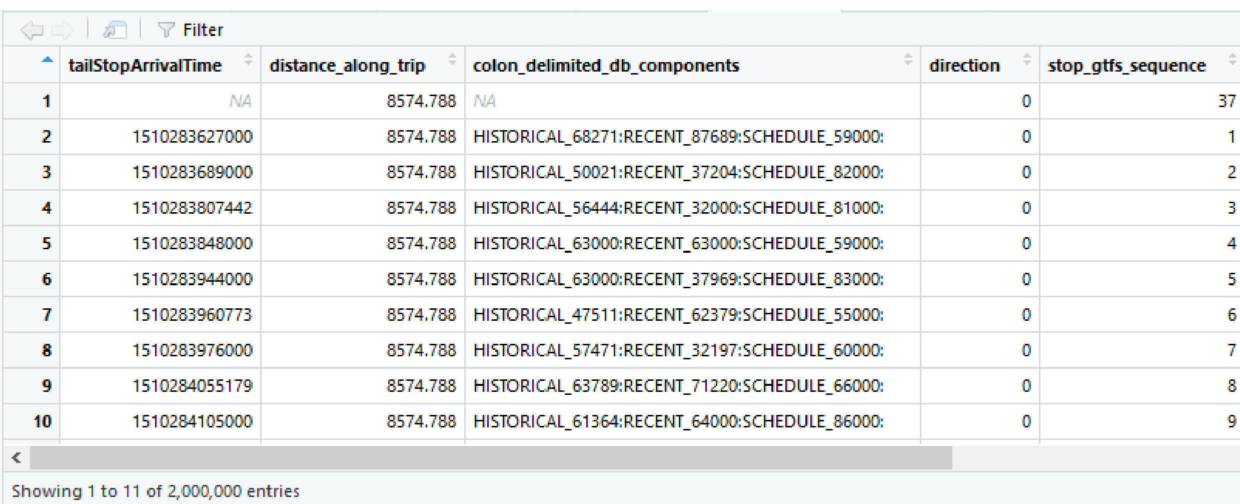


Fig. 2: Data frame window

Once the file is stored as a data frame called Raw_file, this is how it appears with information such as number of attributes or variables (columns) and number of records or observations (rows). Since, in this case, the raw file was divided into chunks having 2 Million records each, this first chunk displays that same number as number of observations.

Just by clicking on the file name in the figure above, the file opens in R-Studio itself, as a data frame, and the following columns can be seen:



Filter

	tailStopArrivalTime	distance_along_trip	colon_delimited_db_components	direction	stop_gtfs_sequence
1	NA	8574.788	NA	0	37
2	1510283627000	8574.788	HISTORICAL_68271:RECENT_87689:SCHEDULE_59000:	0	1
3	1510283689000	8574.788	HISTORICAL_50021:RECENT_37204:SCHEDULE_82000:	0	2
4	1510283807442	8574.788	HISTORICAL_56444:RECENT_32000:SCHEDULE_81000:	0	3
5	1510283848000	8574.788	HISTORICAL_63000:RECENT_63000:SCHEDULE_59000:	0	4
6	1510283944000	8574.788	HISTORICAL_63000:RECENT_37969:SCHEDULE_83000:	0	5
7	1510283960773	8574.788	HISTORICAL_47511:RECENT_62379:SCHEDULE_55000:	0	6
8	1510283976000	8574.788	HISTORICAL_57471:RECENT_32197:SCHEDULE_60000:	0	7
9	1510284055179	8574.788	HISTORICAL_63789:RECENT_71220:SCHEDULE_66000:	0	8
10	1510284105000	8574.788	HISTORICAL_61364:RECENT_64000:SCHEDULE_86000:	0	9

Showing 1 to 11 of 2,000,000 entries

Fig. 3: First five attributes of the data frame.

Before we dive into learning about each column, let us familiarize ourselves, if not already, with the concept of epoch time. The operating system used on the machines is Linux, therefore the timestamps under column names like TailStopArrivalTime, predicted_arrival, time_of_sample are in epoch time.

3.2 What is epoch time?

The Unix epoch (or Unix time or POSIX time or Unix timestamp) is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds (in ISO 8601: 1970-01-01T00:00:00Z). Literally stating the epoch is Unix time 0 (midnight 1/1/1970), but 'epoch' is often used as a synonym for 'Unix time'.

Epoch Time interpretation:

1. In the above figure, considering the third row of column number 1, ‘TailStopArrivalTime’, the value 1510283689000 represents Thursday, November 9, 2017 10:14:49 PM; this is the day, date, and time, respectively for that record.

2. In the figure below, considering the third row of column number 5, time_of_sample, the value 1510283627000 represents Thursday, November 9, 2017 10:13:47 PM.

3.3 Column description:

1. TailStopArrivalTime (**Fig.3 Column 1)**: It is the time taken by the bus to reach the corresponding bus stop. This attribute will further be further used to obtain a new one, referred to as Measured Time or T_m. Units: Epoch timestamp in milliseconds.
2. Distance_along_trip **(Fig.3 Column 2)**: The distance travelled by the bus along the current trip, at the instant where the information is recorded, is displayed in this column. The distance is in meters.
3. Column_delimited_db_components **(Fig.3 Column 3)**: This column contains information on three attributes, namely, Historical, Recent and schedule. These are time components expressed in milliseconds. Historical time represents the time taken by the bus for the past 30 days to reach the bus stop. Recent time represents the time taken for the past 7 days, whereas, Schedule time represents the ideal time that the bus should take to reach the bus stop under preordained conditions.
4. Direction **(Fig.3 Column 4)**: This attribute contains only two unique values, 0 and 1. Each of these two helps to determine whether the bus is running up the route or down the route.
5. Stop_gtfs_sequence **(Fig.3 Column 5)**: The **General Transit Feed Specification** (GTFS) defines a common format for public transportation schedules and associated geographic information. This column answers the question, ‘For which stop in the sequence is this entire record displaying information?’

stop_gtfs_sequence	route	stop_id	predicted_arrival	time_of_sample	vehicle	stop_distance_along_trip
37	MTA NYCT_M15	MTA_404253	0	1510283627000	5889	8438.180
1	MTA NYCT_M15	MTA_401706	1510283642088	1510283627000	5889	8609.666
2	MTA NYCT_M15	MTA_401707	1510283693378	1510283627000	5889	8849.265
3	MTA NYCT_M15	MTA_401708	1510283744956	1510283627000	5889	9087.436
4	MTA NYCT_M15	MTA_401709	1510283807156	1510283627000	5889	9259.072
5	MTA NYCT_M15	MTA_401710	1510283864144	1510283627000	5889	9500.503
6	MTA NYCT_M15	MTA_401711	1510283919100	1510283627000	5889	9662.320
7	MTA NYCT_M15	MTA_401712	1510283966967	1510283627000	5889	9837.473
8	MTA NYCT_M15	MTA_405100	1510284034171	1510283627000	5889	10085.383
9	MTA NYCT_M15	MTA_401715	1510284101517	1510283627000	5889	10412.416

Showing 1 to 11 of 2,000,000 entries

Fig. 4: Next four attributes of the data frame.

6. Route **(Figure 4, column 2)**: Represents the current route on which the bus is running.
7. Stop_id **(Figure 4, Column 3)**: Displays the stop identification number corresponding to the bus stop being shown in the stop_gtfs_sequence column.
8. Predicted_arrival **(Figure 4, Column 4)**: The predicted time of arrival for the bus can be seen under this column. Units: Epoch timestamp in milliseconds. This attribute will further be further used to obtain a new one, referred to as Predicted Time or T_p.
9. Time_of_sample **(Figure 4, Column 5)**: The time instant at which the information (entire row) is recorded. Units: Epoch timestamp in milliseconds. This attribute will further be further used to obtain both Measured Time (T_m) and Predicted Time (T_p).
10. Vehicle (**Figure 4, Column 6**): Contains information about the vehicle number.
11. Stop_distance_along_trip **(Figure 4, Column 7)**: This is the distance between the current stop and the last stop. It denotes the amount of travelling distance the bus is yet to cover. This distance is also represented in meters.

service_date	trip	block	distance_of_trip
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12
1510203600000	MTA NYCT_OH_D7-Weekday-SDon-129000_M15_53	MTA NYCT_OH_D7-Weekday-SDon_E_OH_22620_M15-16	13775.12

Fig. 5: Next four attributes of the data frame.

12. Service date **(Figure 5, Column 1)**: Shows the date on which the information was recorded. Units: Epoch timestamp in milliseconds. However, the time shown is the same for all the records having the same date.
13. Trip **(Figure 5, Column 2)**: This column contains multiple sets of information such as whether the bus is run by MTA BC or MTA NYCT. It contains the name of the depot where the bus originates from, bus number, whether it is a weekday bus service or weekend bus service.
14. Distance of trip **(Figure 5, Column 4):** This is the total distance yet to be covered by the bus. This is also the distance from the current location of the bus to the last stop on the trip.

Note: I have only included 14 out of 18 columns as the remaining ones are never used throughout the analysis and those columns do not contain useful information for achieving our goal.

3.4 Creating new attributes:

Now that we briefly know about each of the attributes, let us focus on the new attributes created using the most essential ones:

Measured Time (T_m) = (tail Stop Arrival Time – Time of Sample)/1000

Predicted Time (T_p) = (predicted arrival – Time of Sample)/1000

Both T_m and T_p have the same units -- time in seconds, after dividing the milliseconds obtained from the difference between the existing attributes.

Residual Time (T_r) = Measured Time – Predicted Time

Residual Time is the error between the measured and predicted time. **Our goal is to minimize this error**. Again, T_r is Residual time in seconds.

Negative Residual Time values represent the buses which arrived before the predicted time and positive ones represent buses which arrived after the predicted time.

For examining the current performance of the prediction system, the mean of the absolute value of the Residual time is used as a metric.

4. Expurgation:

Now that we have the required attributes, let us transform this data into an analyzable form. The question here is, why is it not analyzable? The reason being the following problems:

4.1.1 **Problem 1:** Records showing measured time to be earlier than the time of Sample.

This means, **Measured Time (T_m) = (tail Stop Arrival Time – Time of Sample)/1000,** would result in a negative value.

To remove such errors and to also classify arrival times, we create a new attribute which is a broad categorization in minutes. So, based on the 'breaks' in seconds like '-infinity, 0, 300, 600, 1200, infinity', the measured times are classified as 'Error, 0-5 mins, 5-10 mins, 10-20 mins, 20+ mins' respectively.

-Infinity	0	300	600	1200	Infinity
Error	0-5 mins	5-10 mins	10-20 mins	20+ mins	

Fig. 6: Classification of Measured Time Tm.

In the actual code, more number of breaks have been used. Thus, based on the value of the measured time, it is classified into one of the mentioned categories. This also takes care of the negative T_m values as they are now found under the term 'Error'. Now, we write a command which deletes the records which have been classified as error values.

```
#Classify Measured Time from seconds to minutes and check for errors:
Arrivals$timeperiod = cut(Arrivals$timeToArrival, c(-Inf,0,300,600,1200,1800,3600,Inf),
                          labels=c("err","0--5","5--10","10--20","20--30",">30",">60"))

#Eliminate records classified as Errors:
Arrivals <- subset(Arrivals, timeperiod != "err")
```

Similarly, for T_r, based on the level of granularity desired, the following breaks and categories have been used:

0 secs	60 secs	120 secs	180 secs	240 secs	360 secs
0-1 min	1-2 mins	2-3 mins	3-4 mins	4-6 mins	

Fig. 7: Classification of Residual Time Tr.

This categorization will be used at a later stage where we would require building histograms based on the number of data points present in each of these categories.

4.1.2 Extracting time components:

	colon_delimited_db_components	historical	recent	schedule
6	HISTORICAL_63000:RECENT_37969:SCHEDULE_83000:	63.000	37.969	83
7	HISTORICAL_47511:RECENT_62379:SCHEDULE_55000:	47.511	62.379	55
8	HISTORICAL_57471:RECENT_32197:SCHEDULE_60000:	57.471	32.197	60
9	HISTORICAL_63789:RECENT_71220:SCHEDULE_66000:	63.789	71.220	66
10	HISTORICAL_61364:RECENT_64000:SCHEDULE_86000:	61.364	64.000	86
11	HISTORICAL_73995:RECENT_94029:SCHEDULE_63000:	73.995	94.029	63
12	HISTORICAL_52914:RECENT_89755:SCHEDULE_85000:	52.914	89.755	85
13	HISTORICAL_106270:RECENT_202826:SCHEDULE_75000:	106.270	202.826	75
14	HISTORICAL_32000:RECENT_59406:SCHEDULE_30000:	32.000	59.406	30
15	HISTORICAL_31535:RECENT_31130:SCHEDULE_34000:	31.535	31.130	34
16	HISTORICAL_63000:RECENT_62441:SCHEDULE_60000:	63.000	62.441	60
17	HISTORICAL_46592:RECENT_33000:SCHEDULE_44000:	46.592	33.000	44
18	HISTORICAL_32000:RECENT_33585:SCHEDULE_33000:	32.000	33.585	33
19	HISTORICAL_20930:RECENT_32000:SCHEDULE_30000:	20.930	32.000	30
20	HISTORICAL_43969:RECENT_64000:SCHEDULE_48000:	43.969	64.000	48
21	HISTORICAL_64500:RECENT_77682:SCHEDULE_62000:	64.500	77.682	62

Showing 6 to 22 of 2,000,000 entries

Fig. 8: Time components extracted.

Previously, in the column description section of this document, we learnt about the kind of information that is available in the column named 'colon_db_delimited_components'. Now, the next step is to extract this information and store it as new attributes.

As it can be observed in Fig. the raw information is in the form of a string. In computer programming, a string is a sequence of characters. Therefore, to perform operations on a string, we would require installing a new package called 'stringr'.

```
#install package for performing operations on data type string:
install.packages("stringr")
library(stringr)
```

Now, we begin with the first component, Historical. The code for this component truncates all the characters from ":R" which means, we are now left with whatever appears before ":R". For example, for row number 6 in Fig., we would have

“HISTORICAL_63000”. Further, we need to extract only the numerical values out if this newly obtained string. So, we truncate everything that comes after “L_” and finally store the value in a new column named ‘historical’ as can be seen in the Fig. 8.

```
#Extract time components from attribute 'colon_delimited_db_compnents':

#Sequence in the Nov9 file: HISTORICAL , RECENT, SCHEDULE.

#Based on the sequence of appearance of time components in the string, first we extract Historical time:
Arrivals$historical = sapply(str_split(Arrivals$colon_delimited_db_components, ":R"), function(x){x[1]})
Arrivals$historical = sapply(str_split(Arrivals$historical, "L_"), function(x){x[2]})

#Convert data type string to numeric to perform mathematical operations:
Arrivals$historical <- as.numeric(as.character(Arrivals$historical))
#Divide by 1000 to obtain data in seconds from milliseconds:
Arrivals$historical <- (Arrivals$historical)/1000
```

Similarly, we follow the procedure to get ‘recent’ and ‘schedule’.

```
#Repeat steps for Recent and Schedule:
Arrivals$recent = sapply(str_split(Arrivals$colon_delimited_db_components, ":S"), function(x){x[1]})
Arrivals$recent = sapply(str_split(Arrivals$recent, "T_"), function(x){x[2]})
Arrivals$recent <-as.numeric(as.character(Arrivals$recent))
Arrivals$recent <- (Arrivals$recent)/1000

Arrivals$schedule = sapply(str_split(Arrivals$colon_delimited_db_components, "E_"), function(x){x[2]})
Arrivals$schedule = sapply(str_split(Arrivals$schedule, ":"), function(x){x[1]})
Arrivals$schedule <- as.numeric(as.character(Arrivals$schedule))
Arrivals$schedule <- (Arrivals$schedule)/1000
```

Although these new columns might appear to contain numerical values, indeed they are still in the form of a string since we have extracted a shorter string from a longer one. To be able to perform mathematical operations on these numerical values, we need to convert them to numbers from string. This is achieved using the following line of code:

```
Arrivals$schedule <- as.numeric(as.character(Arrivals$schedule))
```

What is the role played by these time components?

The predictions reported by the cell-phone application Bus-Time, are a function of these three components. As far as just the Predicted Time $\mathbf{T_p}$ and the components are concerned, in terms of regression analysis, $\mathbf{T_p}$ is the dependent variable and the time components – Historical, Recent and Schedule are the independent variables.

Each of these components have a certain coefficient assigned to them and the together with these coefficients and components, the Predicted time is calculated.

0.4*Historical + 0.4*Recent + 0.2*Schedule = $\mathbf{T_p}$

The above set of coefficients is the one which is currently being used for calculating $\mathbf{T_p}$.

It is therefore essential that we have these components separately made accessible to perform the required mathematical operations, also to establish a relationship between these components and the Predicted time, and in turn with the Measured Time $\mathbf{T_m}$.

4.2 **Problem 2:** Presence of Zeroes in time components (Historical and/or Schedule):

We sure have all the time components displayed in seconds but, a significant number of records have one of these three or sometimes two of the three components having value ‘zero’. Ideally, for each of these cases, imputation is the solution, however, for the time being, we have excluded all such groups of records where either of the components is displayed as zero.

	tailStopArrivalTime	distance_along_trip	colon_delimited_db_components	direction	stop_gtfs_sequence	route
1	NA	8574.7877	NA	0	37	MTA NYCT_M15
2	1510283627000	8574.7877	HISTORICAL_68271:RECENT_87689:SCHEDULE_59000:	0	1	MTA NYCT_M15
3	1510283689000	8574.7877	HISTORICAL_50021:RECENT_37204:SCHEDULE_82000:	0	2	MTA NYCT_M15
4	1510283807442	8574.7877	HISTORICAL_56444:RECENT_32000:SCHEDULE_81000:	0	3	MTA NYCT_M15
5	1510283848000	8574.7877	HISTORICAL_63000:RECENT_63000:SCHEDULE_59000:	0	4	MTA NYCT_M15
6	1510283944000	8574.7877	HISTORICAL_63000:RECENT_37969:SCHEDULE_83000:	0	5	MTA NYCT_M15
7	1510283960773	8574.7877	HISTORICAL_47511:RECENT_62379:SCHEDULE_55000:	0	6	MTA NYCT_M15
8	1510283976000	8574.7877	HISTORICAL_57471:RECENT_32197:SCHEDULE_60000:	0	7	MTA NYCT_M15
9	1510284055179	8574.7877	HISTORICAL_63789:RECENT_71220:SCHEDULE_66000:	0	8	MTA NYCT_M15
10	1510284105000	8574.7877	HISTORICAL_61364:RECENT_64000:SCHEDULE_86000:	0	9	MTA NYCT_M15
11	1510284181406	8574.7877	HISTORICAL_73995:RECENT_94029:SCHEDULE_63000:	0	10	MTA NYCT_M15
12	1510284220809	8574.7877	HISTORICAL_52914:RECENT_89755:SCHEDULE_85000:	0	11	MTA NYCT_M15
13	1510284254082	8574.7877	HISTORICAL_106270:RECENT_202826:SCHEDULE_75000:	0	12	MTA NYCT_M15
14	1510284265000	8574.7877	HISTORICAL_32000:RECENT_59406:SCHEDULE_30000:	0	13	MTA NYCT_M15
15	1510284292210	8574.7877	HISTORICAL_31535:RECENT_31130:SCHEDULE_34000:	0	14	MTA NYCT_M15
16	1510284321499	8574.7877	HISTORICAL_63000:RECENT_62441:SCHEDULE_60000:	0	15	MTA NYCT_M15
17	1510284342363	8574.7877	HISTORICAL_46592:RECENT_33000:SCHEDULE_44000:	0	16	MTA NYCT_M15
18	1510284360000	8574.7877	HISTORICAL_32000:RECENT_33585:SCHEDULE_33000:	0	17	MTA NYCT_M15
19	1510284373534	8574.7877	HISTORICAL_20930:RECENT_32000:SCHEDULE_30000:	0	18	MTA NYCT_M15
20	1510284399113	8574.7877	HISTORICAL_43969:RECENT_64000:SCHEDULE_48000:	0	19	MTA NYCT_M15
21	1510284435007	8574.7877	HISTORICAL_64500:RECENT_77682:SCHEDULE_62000:	0	20	MTA NYCT_M15
22	NA	11830.4098	NA	0	44	MTA NYCT_BX1

Showing 1 to 23 of 2,000,000 entries

Fig. 9: Understanding grouping of records.

In the above figure, records from row number 2 up to row number 21, form a group.

For the purpose of filtering data by groups, we need at least one attribute which is constant for the entire group. Column number 2, ‘distance_along_trip’ is one such attribute which has the same value for group of records.

By using package ‘dplyr’ we can achieve the desired grouping and further we filter out the entire group having any of the components with the value 0.

```r
#Eliminate groups of records having one or more time components as Zero:
#for grouping, we require package 'dplyr':
install.packages("dplyr")
library(dplyr)

#In the case of Nov9 file, historical and schedule had zero values in multiple records:
Arrivals<- Arrivals %>%
  group_by(distance_along_trip) %>%
  filter(!any(historical == 0))

Arrivals<- Arrivals %>%
  group_by(distance_along_trip) %>%
  filter(!any(schedule == 0))
```

It is important that all the records satisfy the equation

$$\mathbf{0.4*Historical + 0.4*Recent + 0.2*Schedule = T_p}$$

because, when we have an entire dataset with the dependent variable $\mathbf{T_p}$ being a function of the three independent variables, only then we can test the accuracy of the coefficients.

4.3 **Problem 3:** Buses having bus stops skipped:

	tailStopArrivalTime	stop_gtfs_sequence	stop_id	predicted_arrival	AbsResidual
476	1506699938000	1	MTA_400071	1506699942481	4.481
477	1506700109000	2	MTA_450402	1506700073973	35.027
478	1506700267000	3	MTA_404890	1506700168661	98.339
479	1506700522000	4	MTA_400911	1506700320141	201.859
480	1506700661000	5	MTA_903035	1506700476854	184.146
481	1506700745970	6	MTA_404132	1506700660841	85.129
482	1506700877686	7	MTA_400723	1506700845241	32.445
483	1506701629000	9	MTA_903034	1506701533041	95.959
484	1506701723000	10	MTA_400732	1506701688441	34.559
485	1506701306554	11	MTA_400933	1506701872836	566.282
486	1506701947000	11	MTA_400933	1506701872836	74.164
487	1506702245000	12	MTA_903037	1506702126181	118.819
488	1506702461972	13	MTA_403132	1506702378191	83.781
489	1506702492710	14	MTA_403133	1506702417274	75.436
490	1506702697000	15	MTA_404992	1506702550477	146.523
491	1506703239000	16	MTA_903036	1506702611968	627.032

Showing 475 to 492 of 1,391,847 entries

Fig. 10: Example of a bus-stop skipped.

As it can be seen, there are such cases where an entire row is missing. In such situations, all the predictions given after the skip are inappropriate and are not displayed to the customers even though the data set shows the information. So, it is important that we exclude all such rows after the skip because these will affect the accuracy value when I calculate for the entire data set.

For this, we create a new column that stores the difference between two consecutive stops. So, all the appropriate rows will have the value “1” meaning no skips. But, when a group ends, at suppose stop number 20, the next group begins at stop number 1. Here, the difference will be -19. This can be seen in the following figure:

	tailStopArrivalTime	stop_gtfs_sequence	stop_id	predicted_arrival	AbsResidual	distance_along_trip	diff
2	1506699859000	1	MTA_300551	1506699859965	0.965	9237.49529	1
3	1506700207000	2	MTA_300559	1506700086975	120.025	9237.49529	1
4	1506700367000	3	MTA_300560	1506700247215	119.785	9237.49529	1
5	1506700497000	4	MTA_300563	1506700363161	133.839	9237.49529	1
6	1506700657000	5	MTA_300568	1506700492361	164.639	9237.49529	1
7	1506700754000	6	MTA_307432	1506700616161	137.839	9237.49529	1
8	1506700978000	7	MTA_308003	1506700796561	181.439	9237.49529	1
9	1506701074000	8	MTA_306382	1506700852644	221.356	9237.49529	1
10	1506701132853	9	MTA_300578	1506700895572	237.281	9237.49529	1
11	1506701153931	10	MTA_300579	1506700928671	225.260	9237.49529	1
12	1506701189739	11	MTA_300580	1506700979126	210.613	9237.49529	1
13	1506701202000	12	MTA_308008	1506701009513	192.487	9237.49529	1
14	1506701296000	13	MTA_300582	1506701082275	213.725	9237.49529	1
15	1506701326000	14	MTA_306850	1506701158169	167.831	9237.49529	1
16	1506701390000	15	MTA_306851	1506701209243	180.757	9237.49529	1
17	1506701423000	16	MTA_306852	1506701246499	176.501	9237.49529	1
18	1506701488000	17	MTA_306853	1506701315477	172.523	9237.49529	1
19	1506701552000	18	MTA_306854	1506701378645	173.355	9237.49529	1
20	1506701584000	19	MTA_306881	1506701407769	176.231	9237.49529	1
21	1506701617000	20	MTA_307600	1506701442969	174.031	9237.49529	-19
23	1506700376262	1	MTA_300002	1506699868222	508.040	92.45823	1
24	1506700388765	2	MTA_300003	1506699891714	497.051	92.45823	1

Showing 1 to 23 of 1,304,058 entries

Fig. 11: Numerical difference of bus-stop numbers stored in column ‘diff’.

Therefore, we need to exclude rows with value other than 1 but only within each group.

Forcible assignment of the value “1” to the first row of each group and the word “last” to the last row of each group was done.

Now, whatever value hits up first which is not “1”, from that row, up to the row having value “last” was deleted. This can be seen in the next figure:

	tailStopArrivalTime	stop_gtfs_sequence	stop_id	predicted_arrival	AbsResidual	distance_along_trip	diff
472	1506702930000	18	MTA_306851	1506701913886	1016.114	7109.8575	1
473	1506702958748	19	MTA_306852	1506701951142	1007.606	7109.8575	1
474	1506702987571	20	MTA_306853	1506702020120	967.451	7109.8575	last
476	1506699938000	1	MTA_400071	1506699942481	4.481	30876.9265	1
477	1506700109000	2	MTA_450402	1506700073973	35.027	30876.9265	1
478	1506700267000	3	MTA_404890	1506700168661	98.339	30876.9265	1
479	1506700522000	4	MTA_400911	1506700320141	201.859	30876.9265	1
480	1506700661000	5	MTA_903035	1506700476854	184.146	30876.9265	1
481	1506700745970	6	MTA_404132	1506700660841	85.129	30876.9265	1
482	1506700877686	7	MTA_400723	1506700845241	32.445	30876.9265	2
483	1506701629000	9	MTA_903034	1506701533041	95.959	30876.9265	1
484	1506701723000	10	MTA_400732	1506701688441	34.559	30876.9265	1
485	1506701306554	11	MTA_400933	1506701872836	566.282	30876.9265	0
486	1506701947000	11	MTA_400933	1506701872836	74.164	30876.9265	1
487	1506702245000	12	MTA_903037	1506702126181	118.819	30876.9265	1
488	1506702461972	13	MTA_403132	1506702378191	83.781	30876.9265	1
489	1506702492710	14	MTA_403133	1506702417274	75.436	30876.9265	1
490	1506702697000	15	MTA_404992	1506702550477	146.523	30876.9265	1
491	1506703239000	16	MTA_903036	1506702611968	627.032	30876.9265	last
493	1506699893000	1	MTA_403913	1506699880038	12.962	452.8895	1
494	1506700180000	2	MTA_403855	1506700230838	50.838	452.8895	1
495	1506700371000	3	MTA_403427	1506700552473	181.473	452.8895	1

Showing 448 to 470 of 1,304,058 entries

Fig. 12: Eliminating records after the skipped bus-stop.

The aim has been to delete the highlighted rows so that we can retain only the stops without skips. Of course, there could be other efficient ways of doing this but, at this point, the approach worked fine.

Additional issue: Now we will have 2 kinds of “last” rows, one would be the appropriate one as can be seen in the above figure, there’s a “last” which is not highlighted (row number 474) and there would be another kind which is the inappropriate (highlighted).

For separating these, we again store the difference of ‘stop_gtfs_sequence’ column, because after cleaning highlighted portions, there will be a stop number 7 (row 482) and stop number 16 (row 491) right below it.

We repeat these steps and retain only appropriate “last” rows.

4.4 **Problem 4:** The Predicted Time isn’t always a function of the three components

For every group of records, the very first row has a Predicted time which isn’t the result of the corresponding components. Let us observe this through an example:

	historical	recent	schedule	prediction	diff	stop_gtfs_sequence
21	83.750	102.060	27	46.896	46.896	1
22	54.263	63.000	85	110.801	63.905	2
23	23.826	32.000	70	147.131	36.330	3
24	35.436	32.000	40	182.105	34.974	4
25	31.604	48.866	62	226.693	44.588	5
26	40.154	31.000	44	263.955	37.262	6
27	18.220	32.000	51	294.243	30.288	7
28	93.870	75.516	33	368.597	74.354	8
29	18.053	19.704	57	395.100	26.503	9
30	17.906	17.152	30	415.123	20.023	10

Fig. 13: Storing the numerical difference of prediction times in a new column 'diff'.

The columns historical, recent, schedule, prediction, and diff, all are time in seconds. The column 'diff' is the difference between two consecutive prediction times. In the above figure, if we observe row number 22, the value in the column 'diff' is 63.905, which comes from 110.801 minus 46.896 (prediction in row 22 minus the prediction in row 21). Similarly, the row number 3 for column 'diff' has 36.330, which is 147.131 minus 110.801 (prediction in row 23 minus prediction in row 22).

Now, let us check if the following equation holds true, which is being used to calculate predictions:

0.4*historical + 0.4*recent + 0.2*schedule = $\mathbf{T_p}$

For row number 21:

0.4*83.750 + 0.4*102.060 + 0.2*27 = 79.724

However, the corresponding prediction time is not the same, it is 46.896 seconds.

Let us check for the next couple of rows.

For row number 22:

0.4*54.263 + 0.4*63 + 0.2*85 = 63.905

Yes, the corresponding prediction value indeed is 63.905 seconds.

For row number 23:

0.4*23.826 + 0.4*32 + 0.2*70 = 36.330

Yes, the corresponding prediction value indeed is 36.330 seconds.

If we check for the next rows, every prediction time would be found satisfying the equation. It is always the first row for every group of records that does not follow.

So, why does the first row of every group does not have its prediction time as a function of its corresponding time components?

Let us observe a glimpse of the raw file having the first row of each group with incomplete information.

The purpose of including this record per group is to understand the reason behind the prediction time not following the equation.

The value in the second column- distance_along_trip gives us the exact number of meters traveled by the bus, on that route, at the time when the records were reported (time of sample).

The value in the fourth column- stop_distance_along_trip gives us exact distance in meters at which the bus stop is located on that route from the starting point.

	tailStopArrivalTime	distance_along_trip	stop_gtfs_sequence	stop_distance_along_trip
23	1510283657558	11830.4098	1	12005.1365
24	NA	2135.0072	11	1999.8910
25	1510283659000	2135.0072	1	2328.0302
26	1510283702697	2135.0072	2	2566.8396
27	1510283723000	2135.0072	3	2707.1971
28	1510283787000	2135.0072	4	2919.4315
29	1510283826885	2135.0072	5	3070.2746
30	1510283842093	2135.0072	6	3242.1696

Fig. 14: Demonstrating a portion of the record having values ‘NA’.

Let us have a look at the previous Fig. 13: Storing the numerical difference of prediction times in a new column ‘diff’:

	historical	recent	schedule	prediction	diff	stop_gtfs_sequence
21	83.750	102.060	27	46.896	46.896	1
22	54.263	63.000	85	110.801	63.905	2
23	23.826	32.000	70	147.131	36.330	3
24	35.436	32.000	40	182.105	34.974	4
25	31.604	48.866	62	226.693	44.588	5
26	40.154	31.000	44	263.955	37.262	6
27	18.220	32.000	51	294.243	30.288	7
28	93.870	75.516	33	368.597	74.354	8
29	18.053	19.704	57	395.100	26.503	9
30	17.906	17.152	30	415.123	20.023	10

Fig. 13: Storing the numerical difference of prediction times in a new column ‘diff’.

The prediction of 46.896 seconds is based on the current location of the bus (2135.0072 meters into the trip).

The time components that correspond to it are based on the time required between the next stop and the stop before it (stop number 1 and 11 respectively from fig raw file), which means, the value we obtained from

$0.4*83.750 + 0.4*102.060 + 0.2*27 = 79.724$

was for an estimation from 1999.8910 meters to 2328.0302 meters.

This is the why the first row of every group has the issue of prediction time not being a function of the components.

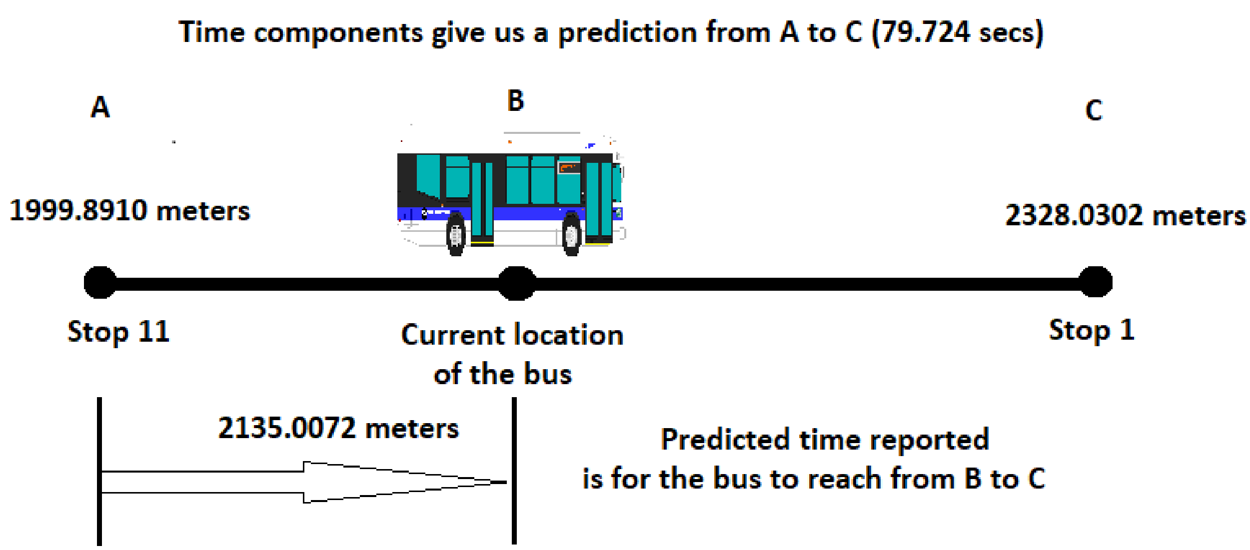


Fig. 15: Visual explanation of the values in the first row for every group of records.

The predicted time and the measured time are originally presented in the raw data as cumulated values.

The time components are presented as the time required between two consecutive stops. Hence, by storing the difference in the prediction times in a new column, we can verify that the components do result in the time stated in the column ‘prediction’.

4.5 Altering the time components:

Let us recollect our goal being to test the efficiency of the current coefficients being used to report predictions. It is therefore important for the prediction time to be a function of these components for all the records present in the dataset.

In the previous sub-section, we learned that after the first row from each group, all the prediction times are indeed a function of the time components. So, if we fix the first row of time components to match with the corresponding predicted time, and go on cumulating the next rows to it, we would have a new set of time components which veritably satisfy our equation.

Now, the question remains, by how much are we required to alter our existing time components?

First, we transform all the piece wise time components (time estimated between two stops) into cumulative values (time required to reach the remaining stops from the current location of the bus).

Again, we need to achieve this for every group of records. The following code helps us to take care of this issue:

```
#Make the time components cumulative:
library(dplyr)
#for historical:
Arrivals<-Arrivals %>%
  group_by(distance_along_trip) %>%
  mutate(histM = cumsum(historical))

#recent
Arrivals<-Arrivals %>%
  group_by(distance_along_trip) %>%
  mutate(recM = cumsum(recent))

#schedule
Arrivals<-Arrivals %>%
  group_by(distance_along_trip) %>%
  mutate(schedM = cumsum(schedule))
```

Now, we have all the three time-components available in a cumulative form. Using the ongoing set of coefficients (0.4, 0.4, 0.2), we calculate the predicted time that these altered components would result into. We create a new column to store the newly calculated predictions.

```
#Calculate new prediction times using the new cumulated time-components and store in a new column "predCal":
Arrivals$predCal <- Arrivals$histM*0.4 + Arrivals$recM*0.4 + Arrivals$schedM*0.2
```

Here lies the answer to our question of by how much should we alter the time components:

```
#Calculate the difference between original and new predictions:
Arrivals$del <- (Arrivals$predCal - Arrivals$prediction)

#Subtract the difference from the cumulated time components and over-write the cumulative time-components:
Arrivals$histM1 <- Arrivals$histM - Arrivals$del

Arrivals$recM1 <- Arrivals$recM - Arrivals$del

Arrivals$schedM1 <- Arrivals$schedM - Arrivals$del
```

We create another column to store the difference between the predictions in the raw data and the new predictions.

```
#Calculate yet another predicted time based on altered time-components:
Arrivals$pCal <- Arrivals$histM1*0.4 + Arrivals$recM1*0.4 + Arrivals$schedM1*0.2
```

The next step is to simply subtract this difference from our cumulated time-components.

Note that the units of all these values are still seconds.

```
#Calculate the difference between original and most recently calculated prediction time:
Arrivals$delta <- abs(Arrivals$pCal - Arrivals$prediction)
```

Finally, we have a prediction time that is a function of our newly obtained cumulative time-components.

To check the validness of these new predictions, we find the difference between these and the raw predictions and store them in a new column. The maximum value in this column is checked as follows:

```
#check for the maximum difference to see if these new predictions are almost equal to original ones:
max(Arrivals$delta)
```

This value isn't significant enough to affect the relationship between the altered cumulative time-components and their corresponding predicted values.

Finally, we have a data set ready to be analyzed.

5. Data segmentation:

The next step is to slice the data by prediction times. We break it down into 7 pieces based on the following time intervals: 0-2 mins, 2-4 mins, 4-6 mins, 6-10 mins, 10-15 mins, 15-20 mins, and 20+ mins. We then separate out the records for the local trips and the express trips. The detailed explanation regarding the separation will be covered in the next section. However, this section describes a common approach used for both local and express buses (after separating).

5.1 Regression:

Linear regression:

- Recent, Historical and Schedule are the three components used for generating predicted times. The approach is to compare the current weights assigned to each of these predictors against the new ones obtained after applying least squared regression.

- The method is to create a matrix of 3 columns (3 predictors) and equate it with a matrix of a single column (measured time).
- The results would be the coefficients for each of the predictors.
- However, these coefficients have constraints. Suppose, x, y, z are the coefficients which are nothing but the weights that were mentioned above, these values should satisfy the equation

  x + y + z = 1.

- So, the matrix would look like, $\mathbf{T_m}$ = x*Historical + y*Recent + z*Schedule
- Therefore, we shall have 3 unknowns (x, y, z) and more than 3 equations (number of equations would be total number of records).

The method of least squares is a standard approach in regression analysis to the approximate solution of over-determined systems, i.e., sets of equations in which there are more equations than unknowns.

We perform least squared regression on each of these pieces (predicted time buckets) and use the following metrics to test the validness of the regression model:

1. The mean of the absolute Residual time of the original data.
2. The mean of the absolute Residual time after applying the optimized weights.
3. The mean of the absolute Residual time after applying the normalized optimized weights.

When we perform a simple linear regression (or any other type of regression analysis), we obtain a line of best fit. The data points usually do not fall *exactly* on this regression equation line; they are scattered around.

A residual is the vertical distance between a data point and the regression line. Each data point has one residual. They are positive if they are above the regression line and negative if they are below the regression line. If the regression line passes through the point, the residual at that point is zero.

Hence, we check for the mean of the absolute residuals.

4. The r-squared value for the Measured time versus the Predicted time.
5. The r-squared value for the Measured time versus the New Predicted time.
6. The r-squared value for the Measured time versus the New Predicted time based on normalized weights.

What does R-squared mean?

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

By definition, R-squared is the percentage of the response variable variation that is explained by a linear model.

Or:

R-squared = Explained variation / Total variation

R-squared is always between 0 and 100%:

- 0% indicates that the model explains none of the variability of the response data around its mean.
- 100% indicates that the model explains all the variability of the response data around its mean.
  In general, the higher the R-squared, the better the model fits our data.

7. Standard deviation of the absolute Residual time.
8. Standard deviation of the new absolute Residual time after applying the optimized weights.
9. Standard deviation of the new absolute Residual time after applying the normalized optimized weights.

Standard deviation is a measure of spread. Specifically, it shows how much of our data is spread out around the mean.

10. Median of the absolute Residual Time of the original data.
11. Median of the absolute Residual Time after applying the optimized weights.
12. Median of the absolute Residual Time after applying the normalized optimized weights.

The median, and particularly the difference between the median and the mean, is useful to characterize how "skewed" the data is.

The median is useful when the dataset may contain extreme outliers. Then, describing the distribution in terms of quartiles (with the median dividing the second from the third quartile) can be more informative than quoting the mean and the standard deviation.

13. The number of data points that lie in the various absolute Residual Time buckets for the original data.
14. The number of data points that lie in the various absolute Residual Time buckets after applying the optimized weights.
    The change in the number of data points between these Residual Time buckets would provide some conclusion regarding the effectiveness of the regression model.

	U	V	W	X	Y	Z	AA	AB	AC
1					ROUTE: MTA NYCT_BX36				
2	Metrics				$T_{predicted\ (min)}$				
3		0 - 2	2-4	4-6	6-10	10-15	15-20	20-30	All
4	Historical optimized	0.575	0.479	0.458	0.437	0.453	0.445	0.434	0.447
5	Recent optimized	0.394	0.386	0.399	0.388	0.362	0.361	0.326	0.349
6	Schedule optimized	0.183	0.189	0.191	0.212	0.225	0.24	0.284	0.247
7	Sum of New Coefficients	1.152	1.054	1.048	1.037	1.041	1.046	1.044	1.043
8	Tr - average (secs)	35.964	47.745	61.644	76.565	101.875	128.847	175.258	113.263
9	Tr_optimized	38.379	49.226	62.793	77.115	102.154	128.07	168.431	111.668
10	Tr_Normalized	36.179	48.121	61.81	76.598	101.86	128.402	173.045	112.712
11	r^2 (T_p vs. T_m)	0.079	0.133	0.109	0.261	0.27	0.198	0.838	0.939
12	r^2 (T_p vs. T_m) optimized	0.089	0.142	0.113	0.263	0.276	0.452	0.841	0.939
13	r^2 (T_p vs. T_m) Normalized	0.089	0.142	0.113	0.263	0.276	0.204	0.841	0.939
14	Sigma of Tr (sec)	94.10	83.90	88.61	90.37	116.51	140.66	169.04	138.87
15	Sigma of Tr (sec) optimized	91.73	81.72	86.17	87.74	110.81	131.39	153.34	129.40
16	Sigma of Tr (sec) Normalized	93.39	82.97	88.06	89.99	115.26	139.57	167.69	137.45
17	Median of Tr (sec)	19.32	33.03	44.00	57.34	75.48	95.31	132.05	71.97
18	Median of Tr (sec) optimized	23.18	35.37	46.66	59.49	78.84	99.45	132.27	74.89
19	Median of Tr (sec) Normalized	19.60	33.37	44.30	57.39	33.37	95.18	130.16	72.08
20		0 - 2	2-4	4-6	6-10	10-15	15-20	20-30	All
21	Total # of Data Points	84360	126533	126711	241237	283672	254233	475418	1592164

	U	V	W	X	Y	Z	AA	AB	AC
20		0 - 2	2-4	4-6	6-10	10-15	15-20	20-30	All
21	Total # of Data Points	84360	126533	126711	241237	283672	254233	475418	1592164
22	# of Residuals in 0-1 min	76905	96909	80389	125220	116528	85222	116899	698072
23	Percentage of data points	91.16	76.59	63.44	51.91	41.08	33.52	24.59	43.84
24	# of Residuals in 1-2 mins	5171	24195	34457	73010	82262	66620	102399	388114
25	Percentage of data points	6.13	19.12	27.19	30.26	29.00	26.20	21.54	24.38
26	# of Residuals in 2-4 mins	856	3767	9458	36153	66881	71440	141671	330226
27	Percentage of data points	1.01	2.98	7.46	14.99	23.58	28.10	29.80	20.74
28	# of Residuals in 4-6 mins	273	527	1052	4284	11717	20063	63786	101702
29	Percentage of data points	0.32	0.42	0.83	1.78	4.13	7.89	13.42	6.39
30	# of Residuals in 6+ mins	1155	1135	1355	2570	6284	10888	50663	74050
31	Percentage of data points	1.37	0.90	1.07	1.07	2.22	4.28	10.66	4.65
32	# of New Residuals in 0-1 min	75038	94170	77782	121485	111545	80002	113361	673867
33	Percentage of data points	88.94974	74.423273	61.3853572	50.3591903	39.3218224	31.4679841	23.8444905	42.32396914
34	# of New Residuals in 1-2 mins	7178	27004	36811	75735	84222	68329	104740	401836
35	Percentage of data points	8.508772	21.3414682	29.0511479	31.3944378	29.6899236	26.8765267	22.0311389	25.23835484
36	# of New Residuals in 2-4 mins	753	3775	9926	38070	71460	76677	147595	349333
37	Percentage of data points	0.892603	2.98341144	7.83357404	15.7811613	25.1910657	30.1601287	31.0453117	21.94076741
38	# of New Residuals in 4-6 mins	265	495	921	3603	11095	20362	68547	105463
39	Percentage of data points	0.31413	0.3912023	0.72685087	1.49355198	3.91120731	8.00918842	14.4182593	6.623877942
40	# of New Residuals in 6+ mins	1126	1089	1271	2344	5350	8863	41175	61665
41	Percentage of data points	1.334756	0.86064505	1.00306998	0.97165858	1.88598099	3.48617213	8.66079955	3.873030668

Fig. 16: Summary table for route MTA NYCT BX36.

Command used for multiple regression:

```
#Perform linear regression using command 'lm':
#We have not included an intercept for this case, hence the presence of a  '+ 0':
ols2 <- lm(buck2$measured~ buck2$histM1 + buck2$recM1 + buck2$schedM1 + 0)
```

5.2.1 Extracting records having Express routes

To separate out buses with express trips we need to create a new column using the information available in the second column- route, in the figure below. The approach is the same as discussed for extracting the time components out of a single string.

stop_gtfs_sequence	route	stop_id	predicted_arrival	time_of_sample	vehicle	stop_distance_along_trip
37	MTA NYCT_M15	MTA_404253	0	1510283627000	5889	8438.180
1	MTA NYCT_M15	MTA_401706	1510283642088	1510283627000	5889	8609.666
2	MTA NYCT_M15	MTA_401707	1510283693378	1510283627000	5889	8849.265
3	MTA NYCT_M15	MTA_401708	1510283744956	1510283627000	5889	9087.436
4	MTA NYCT_M15	MTA_401709	1510283807156	1510283627000	5889	9259.072
5	MTA NYCT_M15	MTA_401710	1510283864144	1510283627000	5889	9500.503
6	MTA NYCT_M15	MTA_401711	1510283919100	1510283627000	5889	9662.320
7	MTA NYCT_M15	MTA_401712	1510283966967	1510283627000	5889	9837.473
8	MTA NYCT_M15	MTA_405100	1510284034171	1510283627000	5889	10085.383
9	MTA NYCT_M15	MTA_401715	1510284101517	1510283627000	5889	10412.416

Showing 1 to 11 of 2,000,000 entries

Fig. 17: Separating express routes.

So, we truncate all the information stored in the column ‘route’ after the occurrence of “_” in the string. We store this obtained piece of information in a new column. For example, we now have smaller strings like, M15, B63, BX2, X1 etc. We are only interested in the express trips; we need to capture only the ones that begin with the letter ‘X’. The following piece of code helps to achieve the desired results:

```
#Separating Express routes from the local ones:
library(stringr)
Arrivals$route1 = sapply(str_split(Arrivals$route, "_"),function(x){x[2]})

Expr = subset(Arrivals, grepl("^X", Arrivals$route1))
```

5.2.2 Problem 5: Retain data before the long jump:

Express buses follow a typical trend:

Stop Number	Distance in meters (Cumulative)	Difference between consecutive stops
1	1000	0
2	2500	1500
3	3500	1000
4	9500	6000
5	11000	1500
6	12000	1000

7	13000	1000

Table: Example of an occurrence of a long jump on an express trip.

- The highlighted row in the table above, is the one where the bus makes a **jump**; the bus runs express between these two stops (stop 3 and stop 4).
- The prediction time given out varies significantly after the jump because increased distance means more uncertainty of arrival time.
- Therefore, for analysis, it is important to only retain stops before the **long jump.**
- The approach is similar to the one used for getting rid of records after a skip was observed in the sequence of bus stops.

- Using the package 'dplyr' for performing data manipulations in each group of records, we store the difference between two consecutive bus stops in a new column.
- The condition for detecting a long jump has been set to 4000 meters.
- We eliminate the records where the difference crosses the number 4000.
- By doing this, we are indeed deleting a record from its respective group which means now we have a skip in the bus-stop numbers.
- We simply repeat the steps for getting rid of all the records in a group after a stop has been skipped.

```
#Store stop distance along trip in a new column
Expr$sdat <- Expr$stop_distance_along_trip

#detect long jumps:
#store the difference between two consecutive stops in a new column 'diff':
Expr$diff <- ave(Expr$sdat, Expr$distance_along_trip, FUN=function(x) c(1, diff(x)))

Expr$diff <- ave(Expr$sdat, Expr$distance_along_trip, FUN=function(x) c(diff(x), "1"))
Expr$diff <- as.numeric(as.character(Expr$diff))

#check number of long jumps:
sum(Expr$diff > 4000)

#Eliminate records with the long jump:
Expr <- subset(Expr, diff < 4000)
```

6.1 Getting the summary table for each route:

Let us revise the steps needed for loading the large raw data file into R-Studio.

- First, we divide the entire file into chunks of equal number of records. Then, we expurgate as required to transform the data into an analyzable form. Further, we make sure that the Predicted time is a function of all the time components- Historical, Recent and Schedule. Before we perform the regression technique, we split the data into two categories- Local trips and Express trips.
- In the case of all the files having local route records, we obtain a subset from each file based on one particular route. Example, in the following figure, we match the keyword "MTA NYCT_M15" from the column named 'route' and, we store all the records that match this keyword in a new file.

- We now have a separate file having all the information for just the route "MTA NYCT_M15". However, we are required to repeat the route extraction for all the chunks obtained.
- Once, we have separate data files obtained from each of these chunks, all we are left to do is binding all these new files into one big file. We must be careful not to let this big file exceed the maximum number of records (2 million) to be able to load it in R-Studio.
- Finally, we have entire data file/(s) with all the information for a single route. To obtain similar files for other routes, we need to repeat the above-mentioned steps from matching the keyword, in this case, a new keyword depending on the desired route, e.g., "MTA NYCT_B63".
- Now, in the case of express trips, we already have files full of records for express routes. Therefore, for different express routes, we need only enter the required keyword e.g., "MTA NYCT_X2" and repeat the steps to obtain a new data file having information about this route.
- Once the new set of files are ready, we can perform linear regression and obtain summary tables for each of these routes. To export the summary tables, into multiple sheets in the same MS Excel file, the following commands can be used:

```
#Store the summary in a new data-frame:
S54_S <- testlocal

#Export summaries to an excel file as multiple sheets:
library(rio)
install.packages("xlsx")

#Set the java environmental variable to the directory having folder 'jre':
Sys.setenv(JAVA_HOME='C:\\Program Files\\Java\\jdk1.8.0_151\\jre')

library(rJava)
library(xlsx)
#Write the first summary to the excel file:
write.xlsx(B6_1S, file="routeSummary.xlsx", sheetName="B6", row.names=FALSE)

#Add the next summary as a new sheet to the same file:
write.xlsx(S54_S, file="routeSummary.xlsx", sheetName="S54_S", append=TRUE, row.names=FALSE)
```

Note: All the files would be exported to the same directory that had been set right in the beginning using the setwd command. If one wishes to export the files to a different location, the path needs to be specified accordingly.

If the summary tables are to be saved to the drive in the form of a .csv file, the following modification does the trick:

```
#exporting as a csv file:
export(s54_S, "s_fifty_four.csv")
```

6.2 Generating Histograms from the obtained information:

Tableau is a software which produces interactive data visualization products focused on business intelligence. While we can still use R-Studio to generate plots and convey meaningful insights through visualization, Tableau offers a better design in terms of user interface and is fairly straightforward which does not make it sound like an extra time killing piece of work.

The data that needs to be visualized is required to be in a tabular form, contingency tables work fine as well. The following code generates one such table that is exported as a .csv file. This file is read into Tableau and just by drag and drop method, we obtain fancy histograms.

```
#Tableau files:

#bucket 2 residuals:
Residual_time_buckets_minutes <- c('0-1','1-2', '2-4', '4-6', '6+', 'total')
RB <- data.frame(Residual_time_buckets_minutes)


b2_0_1 <- sum(buck2$timeRes == "0-1")
b2_1_2 <- sum(buck2$timeRes == "1-2")
b2_2_4 <- sum(buck2$timeRes == "2-4")
b2_4_6 <- sum(buck2$timeRes == "4-6")
b2_6 <- sum(buck2$timeRes == "6+")
b2_total <- nrow(buck2)
RB$Residuals_buck2 <- c(b2_0_1, b2_1_2, b2_2_4, b2_4_6, b2_6,b2_total)

b2_0_1 <- sum(buck2$NewRes == "0-1")
b2_1_2 <- sum(buck2$NewRes == "1-2")
b2_2_4 <- sum(buck2$NewRes == "2-4")
b2_4_6 <- sum(buck2$NewRes == "4-6")
b2_6 <- sum(buck2$NewRes == "6+")
b2_total <- nrow(buck2)
RB$New_Residuals_buck2 <- c(b2_0_1, b2_1_2, b2_2_4, b2_4_6, b2_6,b2_total)
```

In the code shown above, for predicted-time bucket for zero to two minutes, we store the total number of data points under each residual-time bucket to a new column called ‘Residuals_buck2’.

Next, we store total number if data points under each residual-time bucket after applying optimized coefficients. We save this information in a new column ‘New_Residuals_buck2’.

The above code is repeated for all the predicted-time buckets and also for the entire data-set.

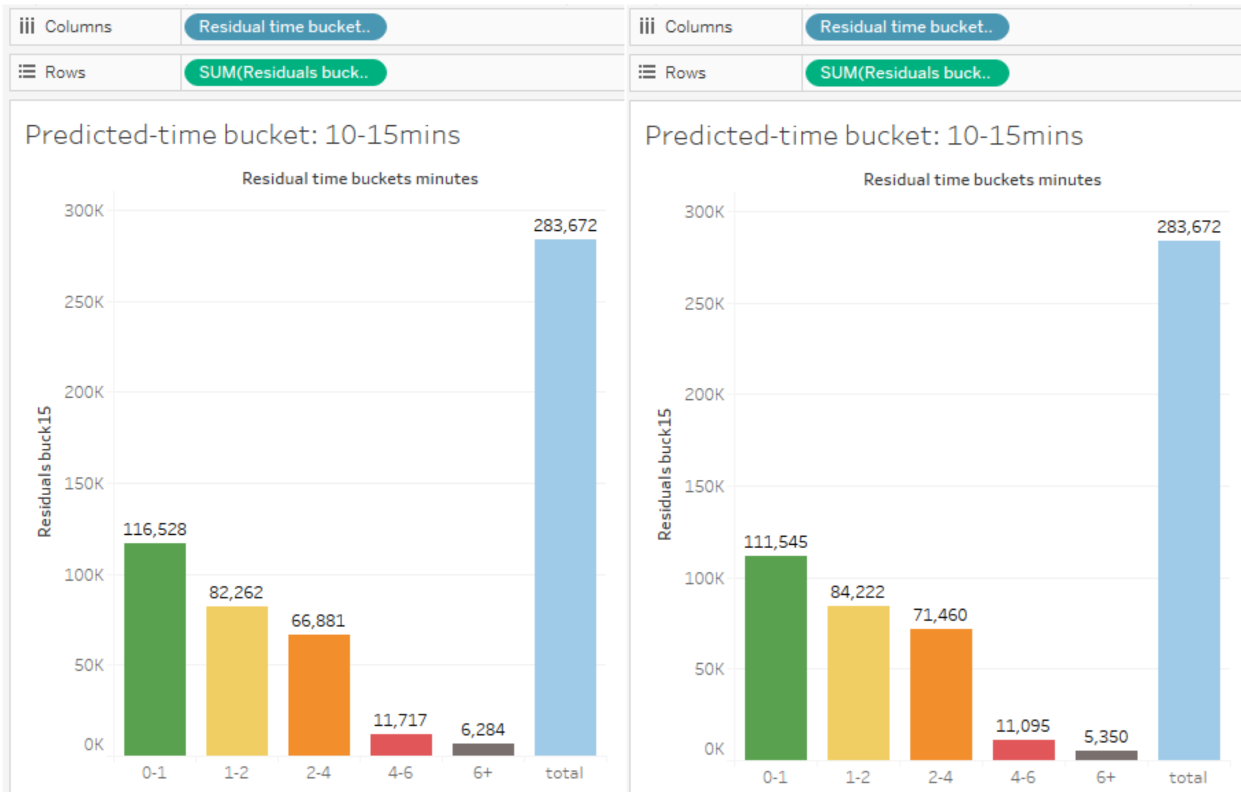


Fig.18: Histogram of number of data points in Residual-time buckets for original (left) and optimized (right) coefficients.

This is an example where one can compare the data points in each Residual-time bucket. There are five such buckets, 0-1, 1-2, 2-4, 4-6, 6+, all in minutes.

A slight decrease in the number of data points can be seen in the last residual-time bucket '6+'. This implies that the optimized coefficients resulted in lesser number of predictions having residual time of more than 6 minutes.

Also, if we observe the first Residual-time bucket '0-1 min', we again see a drop in the number of data points. Let us recall the fact that these residuals are absolute values. The first bucket of 0-1 min contains records where buses have arrived earlier than predicted. The increase in the number of data points in the 1-2 mins bucket, might explain the decrease in the first bucket.

Such histograms can be made for each summary per route.

Future Scope:

Apart from Linear regression, few alternatives could prove efficient. The following links contain materials for such alternatives:

http://erepository.uonbi.ac.ke/bitstream/handle/11295/90013/Onyango_Improving%20Accuracy%20In%20Arrival%20Time%20Prediction%20Using%20Support%20Vector%20Regression.pdf?sequence=3 https://www.svm-tutorial.com/2014/10/support-vector-regression-r/

https://eight2late.wordpress.com/2017/02/07/a-gentle-introduction-to-support-vector-machines-using-r/